Software Design Document

for

Program Review Information System Management

Version 1.0 approved

Prepared by
David, Leanne
McLees, Andrew
Sarenas, Justin
Solis, Ben Jair

For Graduate Studies at California State University, Los Angeles

November 17, 2017

Revision History

Name	Date	Reason For Changes	Version
Initial Version	12/8/2017		1

1. Introduction

1.1 Purpose

The purpose of this document is to describe the implementation details of the Program Review Information System Management (PRISM) as described in the Software System Requirements for Program Review Information System Management. This document describes the entire system and its implementation in detail.

1.2 Document Conventions

This document is formatted according the CSULA CS department's 2017-2018 senior design software design document template.

1.3 Intended Audience and Reading Suggestions

This document is intended for individuals directly involved in the development of PRISM. This includes software developers, team managers, testers, and documentation writers. This document does not need to be read sequentially; users are encouraged to jump to any section they find relevant. Below is a brief overview of each part of the document.

- Part 1 (Introduction)
    - This section offers an overview of PRISM.
- Part 2 (Design Considerations)
    - Readers interested in the considerations accounted for while designing the system should consult this section.
- Part 3 (Architectural Strategies)
    - This section describes the design decisions and strategies used for the organization of the system and its higher-level structures.
- Part 4 (System Architecture)
    - This section provides a high-level overview of how the functionalities and responsibilities of the system were partitioned and then assigned to components.
- Part 5 (Policies and Tactics)
    - This section discusses any relevant political or tactical decisions made and the judgement used behind these decisions.
- Part 6 (Detailed System Design)
    - This section discusses the lower-level details of the system. Each system component is described in detail and may have an accompanying DFD.
- Part 7 (Detailed Lower Level Component Design)
    - This section discusses other lower-level details: components and subcomponents.
- Part 8 (Database Design)
    - This section covers all of the details related to the schema of the database.
- Part 9 (User Interface)
    - This section covers all of the details related to the structure of the graphical user interface (GUI). Readers can view this section for a tentative glimpse of what the final product will look like.
- Part 10 (Requirements Validation and Verification)

    - This section details the requirements and the methods used to test that these requirements are met.
- Part 11 (Glossary)
    - Readers unfamiliar with software engineering terminology should consult this section.
- Part 12 (References)
    - This section includes any additional information which may be helpful to readers.

1.4 System Overview

PRISM is a workflow management tool including an optimized document storage and management system where users will be able to store and download all files related to program reviews

PRISM is a full-stack web application that is accessed via a web browser. The system can be broken up into its back-end and front-end side. The front-end side of the system provides users with an interface for PRISM; functionalities and user input are handled directly here. Whereas the back-end side of the system offers CRUD operations on data in the database.

The following list summarizes the major functions of the system.

- Track and provide an interface to view the progress of each review as it proceeds through the review process
- Store, track the progress of, and allow collaboration on review documents
- Store and automatically source new documents from review document templates
- Store meeting agendas and minutes
- Maintain a calendar of PRS meetings and send e-mail notifications upon changes
- Track which programs are due for review
- Send e-mail notifications upon events relevant to the user

2. Design Considerations

2.1 Assumptions and Dependencies

The following assumptions are being made in the design of PRISM.

- There will be a server capable of running the following software in accordance with the performance requirements of the software:
    - A MEAN stack as specified in section 3 of this document.
    - An e-mail server
- The server will be running a distribution of GNU/Linux
- The server will have sufficiently powerful hardware such that PRISM will not need to undergo extensive optimize to meet performance requirements
- The end users of this software will be limited to:
    - External reviewers
    - CSULA College Deans
    - CSULA Department Chairs
    - PRS members
    - PRS committee chairs/Provost appointees (Administrators)

2.2 General Constraints

End-user Environment

The browsers used by end-users must be supported. This will require additional E2E testing and polyfills in the Angular application to support certain browsers specified in the SRS.

Review Process

The review process has many irregularities and exceptions in timing which must be accounted for in designing time-based parts of the system.

Testing

Testing will be used to ensure reliable and quality of the software. The goal of testing will encourage splitting functionality into smaller modules.

Network Communication

The Angular application will communicate with the server via a REST API. This makes design of the REST API important.

Authentication with CSULA Credentials

As the system must allow users to log in with CSULA credentials, PRISM will need to interface with the university's Shibboleth identity provider server via SAML. This implies certain changes in the user model and the authentication flow.

2.3 Goals and Guidelines

Delivery by the End of the Senior Design Class

The PRS expects the software to be completed in time for it to be used as a part of the 2018-2019 program review process. This will require faster development methods to be utilized.

Emphasis on Simplicity

The user count of the software is likely to never reach 50 concurrent users. This means that performance on the server side will not be a large issue. If faced with situations such as authentication or development of the REST API where access list control or pagination could potentially increase complexity but improve performance or functionality not specified in the requirements, the functionality will only be implemented after all requirements are met. This emphasis is based on the notion that keeping the software simple will speed development.

2.4 Development Methods

Scrum is the software design approach used for this project. This implies that the team meets daily to discuss progress for a short time and that the development of the project proceeds in sprints. For the purposes of this project the sprint length is be two weeks to limit overhead from sprint planning. This method allows the team to make progress and ensure all members of the team are aware of progress being made each day.

3. Architectural Strategies

Use of a particular type of product (programming language, database, library, etc. ...)

PRISM will be using a MEAN stack which is composed of MongoDB, Express, Angular 4, and NodeJS. MongoDB will be interfaced with through Mongoose to assist with data organization and validation. PRISM will also use PrimeNG for UI components for Angular over Bootstrap because it provides rich functionality integrated with Angular.

The primary alternative to a MEAN stack would have been a Java-based stack. The decision to use a MEAN stack was ultimately made because it provides a more responsive user experience, a more flexible software environment, and is widely used in industry.

Reuse of existing software components to implement various parts/features of the system

We will not be reusing existing software components to implement various part/features of the system.

Future plans for extending or enhancing the software

Future plans for PRISM may include but are not limited to:

- A real time collaborative text editor

User interface paradigms (or system input and output models)

PRISM will be accessible through the user's web browser and will take in user input and output the data requested.

Hardware and/or software interface paradigms

PRISM will not have any hardware paradigms. TyperScript is a programming language which may be used to develop JavaScript applications. Javascript supports multiple programming paradigms for ease of development.

Error detection and recovery

PRISM will undergo heavy unit testing to prevent future errors in the system.

Memory management policies

Memory management is not necessary since our server will be able to store all data related to our system.

External databases and/or data storage management and persistence

No external databases or data storage management will be used.

Distributed data or control over a network

Distributed data will not be necessary. Control over a network is given to the administrators of the system which they can access through their CalStateLA user login and password.

Generalized approaches to control

Access is granted by the administrators with the user's CalStateLA credentials.

Concurrency and synchronization

PRISM will need synchronization with the commenting system in the case of two people commenting at the same time.

Communication mechanisms

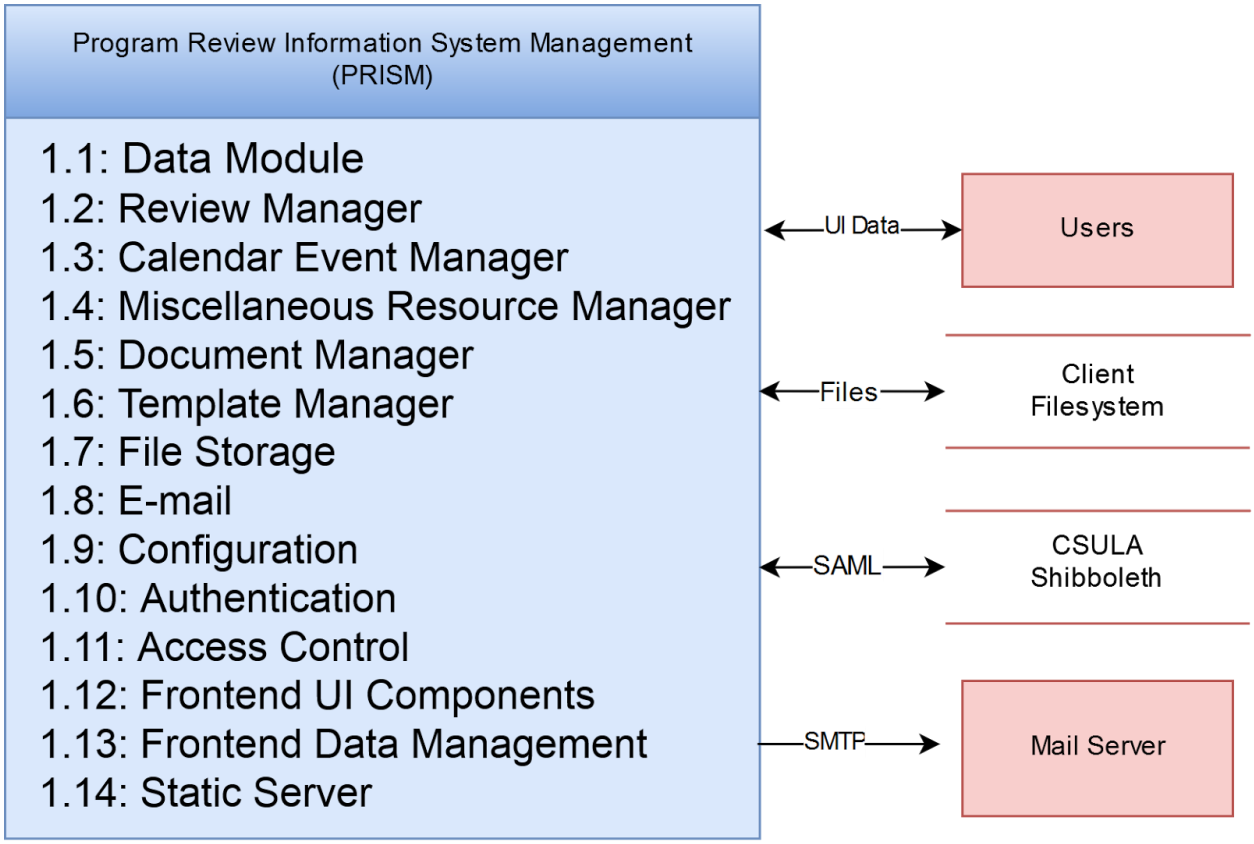
Communication is handled through the web browser.

Management of other resources

There will be no other resources that need to be managed.

4. System Architecture

DFD Level 0

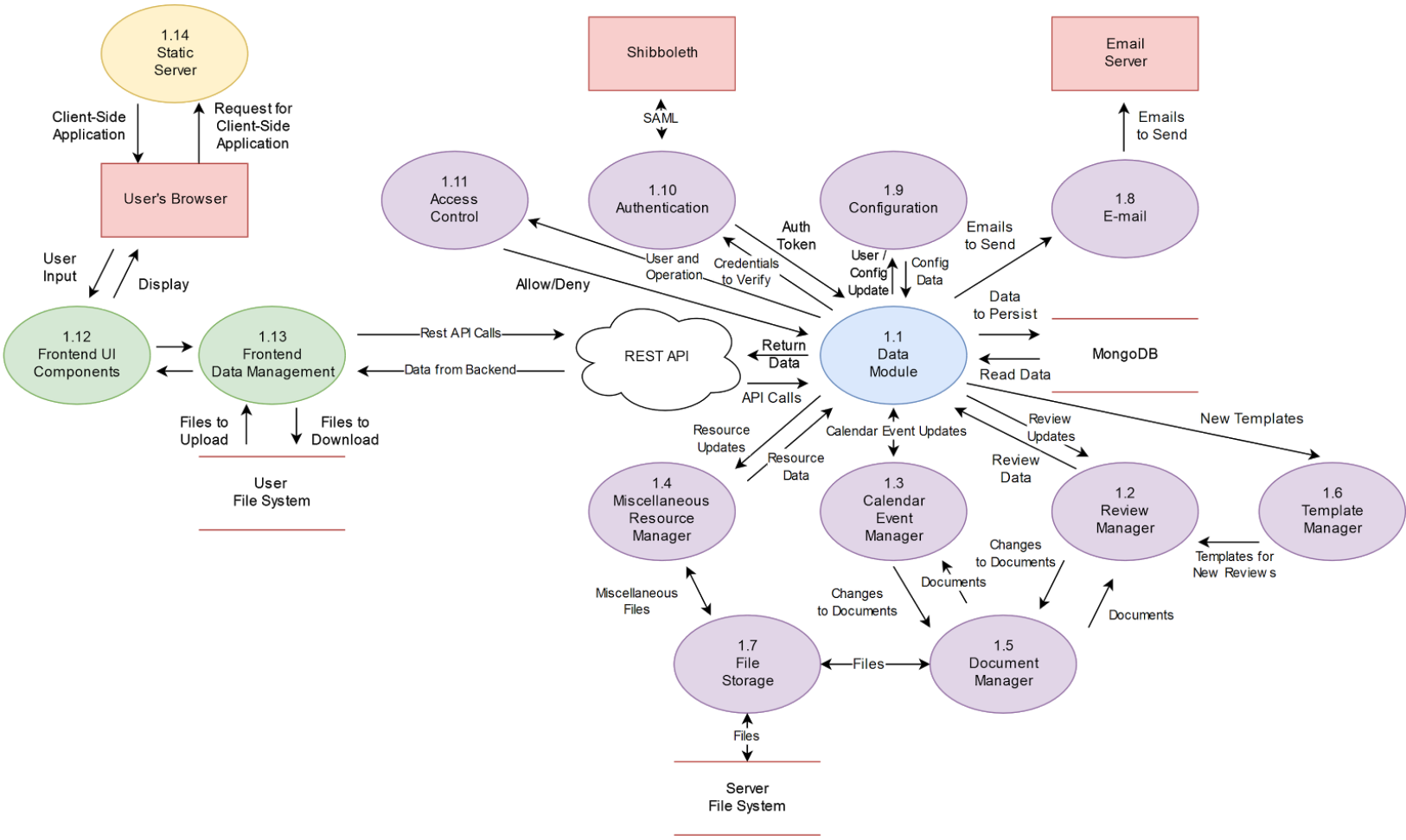


The system was split into modules based on functionality corresponding to categories of requirements (e.g. Review Manager, Calendar Event Manager, Miscellaneous Resource Manager) and functionality that can be separated from the rest of the code base for simplicity (e.g. Authentication, E-mail, etc.). The modules that run on the client side communicate with the server-side modules, and most of the data processing is handled on the server side.

- 4.1.1 Data Module
    - The Data Module handles persisting data to the database and connects all other modules.
- 4.1.2 Review Manager
    - The Review Manager tracks the state of each review.
- 4.1.3 Calendar Event Manager
    - The Calendar Event Manager tracks PRS meetings and their corresponding meeting agendas and minutes.
- 4.1.4 Miscellaneous Resource Manager

    - The Miscellaneous Resource Manager handles files which must be made available to all PRS members (e.g. orientation slideshows).
- 4.1.5 Document Manager
    - The Document Manager handles revisioning and comments for each document stored in PRISM.
- 4.1.6 Template Manager
    - The Template Manager stores and provides templates for the documents involved in reviews.
- 4.1.7 File Storage
    - The File Storage manager handles storage of files on the server filesystem.
- 4.1.8 E-mail
    - The E-mail module handles sending of e-mail to users of the system upon events specified in the requirements occurring.
- 4.1.9 Configuration
    - The Configuration module tracks user and administrator configurations (e.g. e-mail preferences, naming preferences, etc.).
- 4.1.10 Authentication
    - The Authentication modules provides authentication for users both through PRISM directly and through university-provides credentials.
- 4.1.11 Access Control
    - The Access Control module determines whether a user has permission to request a resource or perform an action.
- 4.1.12 Frontend UI Components
    - The Frontend UI Component interact directly with the user, taking actions to execute and form data from the user.
- 4.1.13 Frontend Data Management
    - The Frontend Data Management module deals with data being sent and received on the client side. Building the client side with Angular 4 will allow some processing to be done on the client side to improve the user experience.
- 4.1.14 Static Server
    - The Static Server serves the client-side portion of PRISM to the user over HTTP(S).

DFD Level 1



The overall system was split into three categories of modules: frontend modules that run on the client side, backend modules running on the server, and a static server to serve the client-side code. This distinction was made to simplify development: the frontend and backend can be developed simultaneously with relative ease once the REST API has been specified.

The server was split into a variety of interconnected modules to effectively meet the requirements of the system. The reason for the large number of modules is to meet the single-responsibility principle, which states that each module should have a single responsibility. Having separate modules for different functionality makes development and testing each module easier, and thus each module was chosen to perform a specific set of functions corresponding to the requirements for the system.

5. Policies and Tactics

5.1 Choice of which specific products used

The chosen development tools for which PRISM will be developed in are as follows:

- MEAN stack (Mongoose/MongoDB, Express, Angular 4, NodeJS)
- Atom text editor
- Angular CLI
- Clang-Format
- ESLint
- PrimeNG
- Passport

The main decision made for the software implementation was deciding whether to use Java or JavaScript-based technologies. Ultimately JavaScript was chosen because of its flexibility and the larger range of tools it provides. It is also used more often in current web development trends such as JS frameworks with Angular 4 and its supporting libraries compared to Java servlets.

For writing the code, the text editor that will be used is Atom. Another text editor candidate was neovim but was set aside due to Atom having a smaller learning curve and being simpler to configure. Atom provides the means to download plugins to easily customize the workspace such as clang-format for beautifying and ESLint for linting; this was another factor in choosing Atom as the text editor for developing PRISM.

5.2 Plans for ensuring requirements traceability

Plans for tracking requirements include traversing through the list of requirements and ensuring each case is accounted for.

5.3 Plans for testing the software

Testing the software shall be executed using two testing frameworks−Protractor and Jasmine. Protractor is an end-to-end testing framework that uses test suites for Angular applications. These tests execute while the software is running on the browser and perform scenarios such as a user would. The test code shall be written using Jasmine because of the assertions and functions it provides. Using Protractor and Jasmine seems to be a popular combination which aided in making decisions on which testing frameworks to use.

6. Detailed System Design

Detailed system design will be completed alongside implementation due to the large number of changes that will be imposed by issues of a technical nature.

7. Detailed Lower level Component Design

Detailed lower level component design will be completed alongside implementation due to the large number of changes that will be imposed by technical issues.

8. Database Design

See the PRISM requirements specification document for an overview of the PRISM database design. Detailed database design is not complete; below is a list of models currently implemented. Most notably, the Stage model has not been implemented yet. This list will change as implementation progresses.

The following Mongoose schemas have been implemented. All Mongoose schemas have a MongoDB ObjectId associated with them by default; the schemas below are not an exception. Square brackets (`[]`) indicate arrays in Mongoose schemas.

- College
  - name: String
  - dean: ObjectId
- Comment
  - text: String
  - author: ObjectId
  - creationDate: Date
  - version: ObjectId
- Department
  - name: String
  - college: ObjectId
  - chair: ObjectId
- Document
  - title: String
  - currentRevision: Number
  - allRevision: Array of
    - message: String
    - filePath: String
    - dateUploaded: Date
    - uploader: ObjectId
- Group
  - name: String
  - members: [ObjectId]
- Program
  - name: String
  - department: ObjectId
- Review
  - program: ObjectId
  - startDate: Date
  - finishDate: Date
  - documents: [ObjectId]

- User
    - username: String
    - email: String
    - name:
        - first: String
        - last: String
    - internal: Boolean
    - root: Boolean
    - samlType: String
    - passwordHash: String

9. User Interface

As user interface is a part of implementation, this section will be left to completion after implementation of the user interface.

10. Requirements Validation and Verification

Requirements validation and verification will be conducted after implementation to verify that the implementation meets the requirements.

11. Glossary

Acronyms:

- CSULA - California State University of Los Angeles
- E2E - End-to-End (Testing)
- HTTP - HyperText Transfer Protocol
- HTTPS - HyperText Transfer Protocol over TLS
- MOU - Memorandum of Understanding
- PRISM - Program Review Information System Management
- PRS - Program Review Subcommittee
- SAML - Security Assertion Markup Language
- SMTP - Simple Mail Transfer Protocol as specified in RFC 5321
- TLS - Transport Layer Security

Definitions:

- Document - A single document being produced that contains multiple revisions of its file. (i.e. Self Study, Summary Report, etc.)
- File - Consists of arbitrary data that holds content for a document (i.e. Self Study v1.docx, Self Study v2.docx, etc.)
- Program - A degree program under specific college departments (i.e. Computer Science, Electrical Engineering, etc.)
- Review - A single review process for a department's degree program during a specific year (i.e. CS MS 2018, CS BS 2024, etc.)
- Shibboleth - An implementation of SAML authentication that provides an Identity Provider among other products

12. References

Google JavaScript Style Guide (https://google.github.io/styleguide/jsguide.html)